Robustness of AutoML for Time Series Forecasting in Sensor Networks

Halvari, Tuomas Petteri

IEEE
2021-06

Halvari , T P , Nurminen , J K & Mikkonen , T 2021 , Robustness of AutoML for Time Series Forecasting in Sensor Networks . in 2021 IFIP Networking Conference (IFIP Networking) . IEEE , IFIP Networking Conference , 21/06/2021 . https://doi.org/10.23919/IFIPNetworking52078.2021.9472199

http://hdl.handle.net/10138/333459
https://doi.org/10.23919/IFIPNetworking52078.2021.9472199

acceptedVersion

Robustness of AutoML for Time Series Forecasting in Sensor Networks

Tuomas Halvari
University of Helsinki
Helsinki, Finland
tuomas.halvari@helsinki.fi

Jukka K. Nurminen
University of Helsinki
Helsinki, Finland
jukka.k.nurminen@helsinki.fi

Tommi Mikkonen
University of Helsinki
Helsinki, Finland
tommi.mikkonen@helsinki.fi

***Abstract*—Sensor data collection in IoT networks is sensitive to malfunction of sensors and communications. Hence, it is important that models using the data work in a reasonable way even when there are some, potentially temporary, problems. In this paper, we investigate the robustness of AutoML systems for time series forecasting in sensor networks, using temperature data as example. We experiment with different AutoML systems and study how the resulting models tolerate faults in their input data. The analyzed AutoML systems are Microsoft's Azure AutoML, Intel's Analytics Zoo AutoML, and Facebook's Prophet. As a result, we rank AutoML systems based on their performance with respect to data faults and their severity. In addition, we show how the AutoML generated models differ given the data fault type.**

***Index Terms*—AutoML, time series, forecasting, robustness**

I. Introduction

An increasingly promising approach for ML model creation is the use of automated machine learning (AutoML). In general, AutoML systems include several components of a typical machine learning pipeline like data pre-processing, model selection and hyperparameter optimization. AutoML systems are typically used for tasks like classification and regression. However recently some AutoML systems have started to support more focused tasks like time series forecasting, which in many cases builds on regressors.

Unlike laboratory experiments, real applications operate in different and sometimes unexpected conditions. Therefore, robustness is an important application attribute. A robust ML model tolerates problems with respect to sensor readings, and so on, but still produces good quality forecasts.

In this paper, we focus on the task of time series forecasting. We study how tolerant some AutoML systems are to data faults by controlling the type and the severeness of the data faults injected to the training data and benchmarking them by comparing the forecast to the real outcome. We use easy-to-use AutoML systems, which require only the data, the task, and a time limit for model search and optimization. The key contributions of our paper are: (i) Implement a test setup to study the effect of sensor network faults to the performance of different AutoML time series forecasting systems (Section III). (ii) Measure and compare how some popular AutoML system perform with typical network faults (Section IV).

II. Data faults in sensor timeseries

Time series forecasting is the use of a model, based on a series of data points listed in time order, to predict future values based on previously observed values [1].

Faults in time series data can arise from multiple reasons. Common sources of faults are misbehaving sensors, due to hardware fault, wear, misconfiguration, or miscalibration for instance, and misbehaving networking. Different underlying reasons will be visible in different ways in the time series data. Both the type and the scale of faults can vary.

A taxonomy of sensor network faults is missing. Sharma et al. [2] identified three sensor network faults (single-sample spikes, longer duration noisy readings, and anomalous constant offset readings). Mahapatro and Khilar [3] classified sensor network faults as crash, omission, timing, incorrect computation, fail-stop, authenticated Byzantine, and Byzantine.

We use the classification of Ni et al. [4], which identifies nine different categories of sensor network faults: outliers, spikes, stuck-at fault, high noise or variance, calibration fault, failing connection or HW, low battery, environment our of range, and clipping. We investigated outliers, stuck-at, and high noise cases. In this paper we only report the results of stuck-at case. In case of a **stuck-at fault**, the time series value is fixed and experiences zero variance over a longer period of time. In our tests data fault level controls the position of said fault.

III. Methodology

We studied three systems with different operating principles. **Intel's Analytics Zoo** with the Bayes Recipe constructs a neural network with Long Short-Term Memory (LSTM) layers. **Microsoft's Azure AutoML** constructs voting regressor ensemble models using base regressors mostly from the sklearn library [1]. **Facebook's Prophet** constructs an additive regression model that is Bayesian-influenced [5]. It uses Stan [6] for the underlying calculations.

The experimental task is forecasting the last 20% of the time series dataset, based on temperature sensor data. Given the AutoML system and some version of the training data, each AutoML system is given a time limit in which to find and train the optimal forecasting model. The time limits used

Annex to ISBN 978-3-903176-39-3 ©2021 IFIP

[1] https://scikit-learn.org/stable/

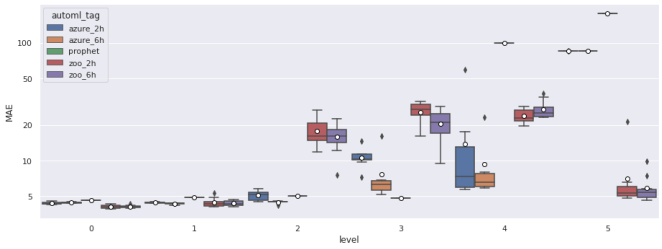
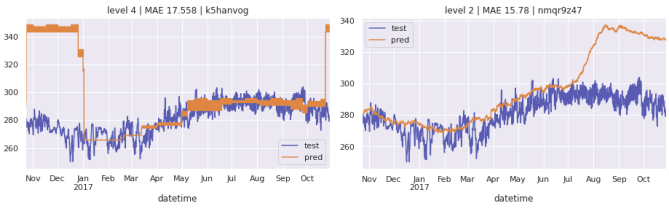


Fig. 1: The score distributions for the benchmarks with stuck-at fault in the training data. Logarithmic scale is used.



(a) Azure AutoML at level 4. (b) Analytics Zoo at level 2.

Fig. 2: Examples of bad forecasts with stuck-at fault.

in our benchmarks are 2 and 6 hours. We present the score distributions of at least 10 runs for the 2 hour benchmarks and at least 5 runs for the 6 hour benchmarks in Section IV. Before running any benchmarks different versions of the training data were generated for each combination of data fault type and data fault level, using a fixed seed. As the dataset we use real-world temperature data from the city of Montreal. Raw data was acquired from the OpenWeatherMap website [2]. For comparing the forecasts to the actual testing data we used out-of-sample mean absolute error (MAE). All benchmarks were run locally on CSC's Puhti cluster [3] using a total of 40 CPUs and 192 GB of RAM.

We generated artificial faults to the dataset in a controlled fashion. With stuck-at fault, at level 1, the stuck-at area is the first fifth of the training data, and so on, until at level 5, the stuck-at area is is the last fifth of the training data. Since the size of the training data is not divisible by 5, the last 3 datapoints retain their original values.

To enable fair comparison, AutoML systems were only given the number of datapoints to forecast, the time limit, and the training data. Other parameters were set to default or "auto" where possible. For Analytics Zoo and Azure we made separate experiments with 2h and 6h time limits. Prophet's execution took only a few minutes and required no time limit.

IV. Results

Full results are available at GitHub repository [4]. The MAE scores of different test cases are presented in Figure 1. In the boxplots, the scores over multiple runs for each AutoML system are presented as distributions for each discrete data

[2]https://openweathermap.org/

[3]https://docs.csc.fi/computing/overview/

[4]https://github.com/thalvari/AutoML_Timeseries

type	MAE	batch_size	dropout_1	dropout_2	lr	lstm_1_units	lstm_2_units	HOUR	DAY	MONTH	WEEKDAY	IS_AWAKE	IS_BUSY_HOURS	IS_WEEKEND
clean	4.13	32	0.2	0.2	0.01	106	125	1	1	1	1	0	0	1
clean	4.131	32	0.2	0.2	0.002022338076571248	73	111	1	1	1	0	0	1	0
clean	4.093	32	0.2	0.2	0.01	103	128	1	1	1	1	1	0	0
clean	4.045	32	0.2	0.2	0.01	112	128	1	1	1	1	1	0	1
clean	4.008	32	0.2	0.2	0.001	128	117	1	1	1	1	0	1	0
stuck-at_5	21.511	74	0.2105276816068324	0.2220144611739566	0.009689433918060479	104	126	1	1	1	1	0	0	1
stuck-at_5	5.981	39	0.2556922424837283565	0.2533859649614515	0.009881574694181088	121	77	1	1	1	1	0	0	1
stuck-at_5	5.373	32	0.2000000000011669915	0.2	0.01	101	49	1	1	1	1	1	1	0
stuck-at_5	5.197	41	0.2	0.2000000141964796	0.01	125	74	1	1	1	1	0	0	0
stuck-at_5	4.818	42	0.2	0.2	0.009999985429257539	91	109	1	1	1	1	0	0	0

TABLE I: Models generated by Analytics Zoo in 2h runs.

type	MAE	step_0	step_1	step_2	step_3	step_4	step_5	step_6
clean	4.352	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	LassoLars
clean	4.372	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	LassoLars	-
clean	4.384	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	LassoLars
clean	4.51	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	LassoLars	-
clean	4.428	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	LassoLars	-
stuck-at_4	9.141	DecisionTreeRegressor	DecisionTreeRegressor	-	-	-	-	-
stuck-at_4	6.869	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	-	-	-	-
stuck-at_4	7.728	DecisionTreeRegressor	DecisionTreeRegressor	-	-	-	-	-
stuck-at_4	59.628	DecisionTreeRegressor	DecisionTreeRegressor	-	-	-	-	-
stuck-at_4	17.558	DecisionTreeRegressor	DecisionTreeRegressor	DecisionTreeRegressor	-	-	-	-

TABLE II: Models generated by Azure AutoML in 2h runs.

fault level, using the MAE metric described above. In the figures, the means for each distribution are represented as white circles.

For both Analytics Zoo and Azure AutoML, the generated models for 5 runs in the 2h benchmark can be seen in Tables I and II, respectively. In both tables, the first two columns tell us what type of training data was used and the MAE score of the run. The following columns are all the relevant information scraped from the log file of each run. Looking at the column names in Figure I, we can see that, with the Bayes recipe that we are using, Analytics Zoo always builds Deep Learning models with two LSTM layers with the corresponding Dropout layers. We can also see that Analytics Zoo constructs some new features from the data, related to periodicity and influence of e.g. weekends. Looking at the column names in Figure II we can see that, Azure AutoML constructs ensemble models, and the columns represent the base regressors, whose results are averaged by the voting regressor.

A. Clean data

The performance comparison for the tested AutoML systems for clean training data can be seen in Figure 1, when looking at the distributions at level 0. Looking at the mean values, at level 0, Analytics Zoo is the winner followed by Azure AutoML as the close second and Prophet as the close third. Looking at the score distributions at level 0 in the same figure, there was little variance in the scores for all of the benchmarked systems. Furthermore, increasing the time limit to 6 hours seems to have a minimal impact to the performance of both Analytics Zoo and Azure AutoML.

The generated models for all runs with Analytics Zoo and clean training data can be seen in the corresponding rows of Table I. Looking at rows of type clean, the first LSTM layer seems to typically have less units compared to the second layer. Also the Dropout probabilities and the learning rate do not change much. In addition three of the features are only used in half of the runs. The generated models for all runs with Azure AutoML and clean training data can be seen in the corresponding rows of Table II. Looking at rows of type clean, it seems that Azure AutoML prefers to use Decision Tree and Lasso Least Angle regressors as the base regressors.

B. Stuck-at faults

Looking at Figure 1, at level 1, Analytics Zoo and Azure AutoML perform somewhat similarly, while Prophet ends

up trailing the two in performance. It is also interesting that different AutoML systems seem to have different drop-off points at mid levels. Analytics Zoo's performance drops clearly at level 2, Prophet's performance drops clearly at level 4 and Azure AutoML's performance drops clearly at the final level, although there are also some bad forecasts at level 4.

A bad forecast for Azure AutoML at level 4 is presented in Figure 2a. A bad forecast for Analytics Zoo at level 2 is presented in Figure 2b. Comparing the two examples, while Analytic Zoo's forecast can start drifting off at some point, Azure AutoML's forecast experiences smaller stuck-at periods within the forecast. While all systems are fine with the stuck-at area being in the beginning of the data, Analytics Zoo is the only system that can handle the stuck-at area being at the end of the training data. So Analytics Zoo's performance actually improves drastically going from level 4 to level 5. As can be seen in Figure 1, Azure AutoML had low variance in its results, except at level 4, while Analytics Zoo had a moderate amount of variance in its scores at mid and high levels. Looking at the same Figure both Azure AutoML and Analytics Zoo gained a bit performance and a reduced variance at mid levels when time limit was increased to 6 hours.

Comparing the rows with type stuck-at in Table I to the rows with type clean, we can see that both batch size and dropout percentages are higher on average and that the second LSTM layers have clearly less units on average. In fact, in many cases the second LSTM layer has less units compared to the first layer. Looking at Figure 1 we can see that at level 5 Azure AutoML fails to give a reasonable forecast. Judging from Table II, at level 4 Azure AutoML prefers to use Decision Tree regressors as the base regressors, though fewer than with clean training data.

V. Discussion

While all compared systems provide similar features, they all are clearly different, making it difficult to compare them technically in terms of other than input, output, and performance. With LSTM models, deeper layers are used capture more high level or abstract features from the data [7], which in case of time series would correspond to longer term trends. Analytics Zoo seems to exclusively generate two-layer LSTM models (Table I). So, it can be said that the second layer is used to capture more general trends in the training data, while the first layer is used to describe the more immediate changes. On the other hand the width of each LSTM layer, namely the number of units on a layer, determines how strongly the relationships between the inputs to a specific layer are taken into account by the model [8]. So, too few units on a layer can lead to underfitting, while too many units can lead to overfitting. In addition, both LSTM layers have an associated Dropout layer in our generated models. The Dropout layer helps to reduce overfitting by excluding some units from updates.

Thus if the number of units on the first layer is low, not many units are required to describe the relationships between nearby datapoints in the time series. Finally, if the number of units on the second layer is low, it could be said that the generated model tries to only capture the simpler relationships between the longer term trends in the data. As shown in Table I, this is the case with training data that has a stuck-at fault that breaks the longer term trends.

With stuck-at fault Azure AutoML prefers using decision tree regressors as base regressors. An example forecast using such model is presented in Figure 2a. One reason why decision tree regressors work with this type of data is that they do not try to fit a single model for the whole data, but instead use recursive partitioning and local fitting of simple models.

It was interesting to see that some systems performed well with certain data fault type and level combinations, but did not do so well with another setup. Also the generated models were quite different given the type of fault in the training data. Therefore, for a certain problem, one of the systems might be much more effective than others, but in general this can be found out by running suitable tests. For cases where training must be fast, Prophet is clearly the best, with its forecasting performance comparable with other tested systems.

Better understanding of the occurrence of different faults in sensor networks and robust machine learning solutions to deal with them still require further research.

References

[1] C. Chatfield, *Time-series forecasting*. CRC press, 2000.

[2] A. B. Sharma, L. Golubchik, and R. Govindan, "Sensor faults: Detection methods and prevalence in real-world datasets," *ACM Transactions on Sensor Networks*, vol. 6, no. 3, pp. 1–39, 2010.

[3] A. Mahapatro and P. M. Khilar, "Fault Diagnosis in Wireless Sensor Networks: A Survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2000–2026, 2013.

[4] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava, "Sensor network data fault types," *ACM Transactions on Sensor Networks*, vol. 5, no. 3, pp. 1–29, 2009.

[5] S. J. Taylor and B. Letham, "Forecasting at scale," *PeerJ Preprints*, vol. 5, p. e3190v2, Sep. 2017.

[6] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A Probabilistic Programming Language," *Journal of Statistical Software, Articles*, vol. 76, no. 1, pp. 1–32, 2017.

[7] M. Hermans and B. Schrauwen, "Training and Analyzing Deep Recurrent Neural Networks," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*. Curran Associates Inc., 2013, pp. 190–198.

[8] S. Chakraborty, J. Banik, S. Addhya, and D. Chatterjee, "Study of Dependency on number of LSTM units for Character based Text Generation models," in *Proceedings of the 2020 International Conference on Computer Science, Engineering and Applications*, 2020, pp. 1–5.